

A Distributed Message-Optimal Assignment on Rings*

Gianluca De Marco[†] Mauro Leoncini[‡] Manuela Montangero[§]

Abstract

Consider a set of items and a set of m colors, where each item is associated to one color. Consider also n computational agents connected by a ring. Each agent holds a subset of the items and items of the same color can be held by different agents. We analyze the problem of distributively assigning colors to agents in such a way that (a) each color is assigned to one agent only and (b) the number of different colors assigned to each agent is minimum. Since any color assignment requires the items be distributed according to it (*e.g.* all items of the same color are to be held by only one agent), we define the cost of a color assignment as the amount of items that need to be moved, given an initial allocation. We first show that any distributed algorithm for this problem requires a message complexity of $\Omega(n \cdot m)$ and then we exhibit an optimal message complexity algorithm for synchronous rings that in polynomial time determines a color assignment with cost at most three times the optimal. We also discuss solutions for the asynchronous setting. Finally, we show how to get a better cost solution at the expenses of either the message or the time complexity.

keywords: algorithms; distributed computing; leader election; ring.

*A preliminary version of this paper has been presented at IPDPS06.

[†]Dipartimento di Informatica, Università di Salerno, Italy. e-mail: demarco@dia.unisa.it

[‡]Dipartimento di Scienze Fisiche, Informatiche e Matematiche, Università di Modena e Reggio Emilia, Italy, and Istituto di Informatica e Telematiche, CNR Pisa, Italy. e-mail: leoncini@unimore.it

[§]Dipartimento di Scienze Fisiche, Informatiche e Matematiche, Università di Modena e Reggio Emilia, Italy, and Istituto di Informatica e Telematiche, CNR Pisa, Italy. e-mail: manuela.montangero@unimore.it

1 Introduction

We consider the following problem. We are given a set of computational agents connected by a (physical or logical) ring¹, and a set of items, each associated to one color from a given set. Initially each agent holds a set of items and items with the same color may be held by different agents (*e.g.* see Fig 1.(a)). We wish the agents to agree on an assignment of colors to agents in such a way that each color is assigned to one agent only and that the maximum over all agents of the number of different colors assigned to the same agent is minimum. We call this a *balanced assignment*: Fig 1.(b) and Fig 1.(c) show two possible balanced assignments. Among all such assignments, we seek the one that minimizes the total number of items that agents have to collect from other agents in order to satisfy the constraints. For example, agent a_0 in Fig 1.(b) is assigned colors ∇ and \spadesuit , and therefore needs just to collect four items colored ∇ , since no other agent has items colored \spadesuit .

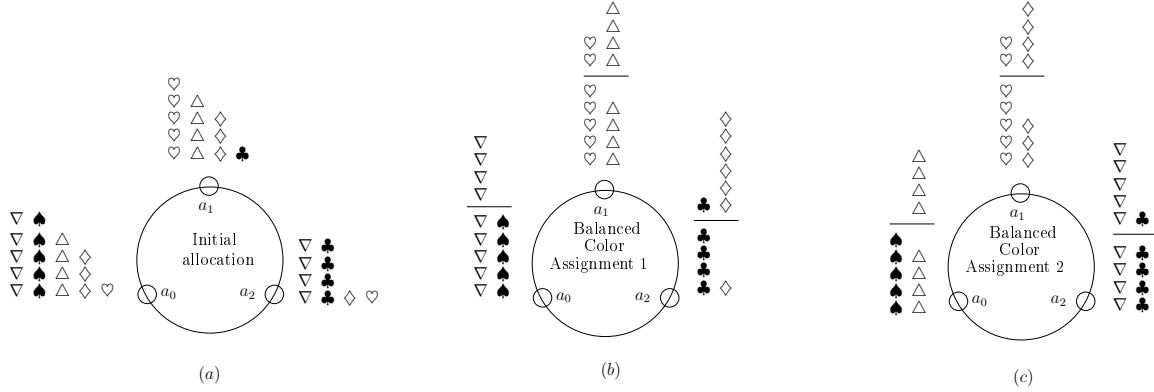


Figure 1: Three agents: a_0, a_1, a_2 , and six colors: $\nabla, \diamond, \heartsuit, \triangle, \spadesuit, \clubsuit$. (a) is the initial allocation, while (b) and (c) are two possible balanced color assignments. Items above the line are those that the agent collects from the others. Therefore their total number is the cost of the assignment. The assignment in (b) costs $(4 \times \nabla) + (2 \times \heartsuit + 4 \times \triangle) + (1 \times \clubsuit + 6 \times \diamond) = 17$ items, while the assignment in (c) costs $(4 \times \triangle) + (2 \times \heartsuit + 4 \times \diamond) + (5 \times \nabla + 1 \times \clubsuit) = 16$ items.

The problem can be formalized as follows. Let $\mathcal{A} = \{a_0, \dots, a_{n-1}\}$ be a set of n agents connected by a ring and let $\mathcal{C} = \{c_0, \dots, c_{m-1}\}$ be a set of m colors. Let $Q_{j,i} \geq 0$ be the number of items with color c_j initially held by agent a_i , for every $j = 0, \dots, m-1$, and for every $i = 0, \dots, n-1$.

Definition 1 (Balanced Coloring). *A Balanced Coloring is an assignment $\pi : \{0, \dots, m-1\} \rightarrow \{0, \dots, n-1\}$ of the m colors to the n agents in such a way that:*

- for every color c_j , there is at least one agent a_i such that $\pi(j) = i$;

¹An important example of logical architecture is given by the set of ring shaped nodes of a Distributed Hash Table.

- for every agent a_i , $\lfloor \frac{m}{n} \rfloor \leq |\{c_j \mid \pi(j) = i\}| \leq \lceil \frac{m}{n} \rceil$; i.e., the number of color assigned to agents has to be balanced. In particular, $\lfloor \frac{m}{n} \rfloor$ colors are assigned to $[(\lfloor \frac{m}{n} \rfloor + 1)n - m]$ agents, and $\lfloor \frac{m}{n} \rfloor + 1$ colors to the remaining $(m - \lfloor \frac{m}{n} \rfloor n)$ agents.

Any Balanced Coloring then assigns almost the same number of colors to each agent, and when m is a multiple of n , then each agent is assigned exactly the same number of colors.

Definition 2 (Distributed Balanced Color Assignment Problem). *The Distributed Balanced Color Assignment Problem aims at distributively finding a Balanced Coloring of minimum cost, where the cost of a Balanced Coloring $\pi : \{0, \dots, m-1\} \rightarrow \{0, \dots, n-1\}$ is defined as*

$$Cost(\pi) = \sum_{j=0}^{m-1} \sum_{\substack{i=0, \\ i \neq \pi(j)}}^{n-1} Q_{j,i}. \quad (1)$$

The cost of the optimal assignment will be denoted by $Cost_{OPT}$. The *approximation ratio* of a sub-optimal algorithm **A** is the quantity $\frac{Cost_{\mathbf{A}}}{Cost_{OPT}}$, where $Cost_{\mathbf{A}}$ is the cost of the solution computed by **A**.

Motivations. The scenario defined above may arise in many practical situations in which a set of agents independently search a common space (distributed crawlers, sensor networks, peer-to-peer agents, etc) and then have to reorganize the retrieved data (items) according to a given classification (colors), see for example [14, 19, 21]. In these cases, determining a distributed balanced color assignment may guarantee specialization by category with maximal use of data stored in local memory or balanced computational load of agents minimizing the communication among agents.

A similar scenario may also arise in computational economics [2]. The distributed balanced color assignment formalizes a combinatorial auction problem where agents are the bidders and colors represent auction objects. The number of items that an agent holds for each color can be interpreted as a measure of *desire* for certain objects (colors). Balancing the number of colors per agent and minimizing the cost guarantees the maximum bidders satisfaction.

The model. We assume that the agents in $\mathcal{A} = \{a_0, \dots, a_{n-1}\}$ are connected by a ring: agent a_i can communicate only with its two neighbors $a_{(i+1) \bmod n}$ (clockwise) and $a_{(i-1) \bmod n}$ (anti-clockwise). We assume that each agent knows n (the number of agents), and \mathcal{C} (the set of colors). Each agent a_i is able to compute $p_i = \max_{0 \leq j \leq m-1} Q_{j,i}$ independently, i.e., the maximum number of items it stores having the same color, while $p = \max_{0 \leq i \leq n-1} p_i$ is unknown to the agents.

We will consider both synchronous and asynchronous rings, always specifying which case we are working with or if results hold for both models.

For synchronous and asynchronous rings, we measure message complexity in the standard way (cf. [16, 18]), *i.e.*, we assume that messages of bit length at most $c \log n$, for some constant c (called *basic messages*), can be transmitted at unit cost. One message can carry at most a constant number of agent ID's. Non basic messages of length L are allowed, and we charge a cost $c' \lceil L / \log n \rceil$ for their transmission, for some constant c' .

For what concerns time complexity, in the synchronous case we assume that agents have access to a global clock and that the distributed computation proceeds in rounds. In each round any agent can check if it has received a message (sent in the previous round), make some local computation, and finally send a message. In the asynchronous case agents don't have access to a global clock, but the distributed computation is *event driven* ("upon receiving message α , take action β "). A message sent from one agent to another will arrive in a finite but *unbounded* amount of time.

Throughout the paper we will use the generic term *time unit* to designate the time needed for a message to traverse a link both in the synchronous and asynchronous case: for the synchronous case a time unit (also called round or time slot) is the time elapsed between two consecutive ticks of the clock; for the asynchronous setting a time unit can be any bounded finite amount of time. Nevertheless, in both cases the time complexity can be simply measured as the number of time units needed to complete the algorithm's execution.

Outline of the results. The goal of this paper is to analyze the efficiency with which we can solve the Distributed Balanced Color Assignment problem. In Section 2 we discuss some related problems and show the equivalence with the so called weighted β -assignment problem in a centralized setting [4]. We also show that a brute force approach that first gathers all information at one agent, then computes the solution locally and finally broadcasts it, has a high message complexity of $O(mn^2 \log p / \log n)$. Fortunately, we can do better than this. In Section 3 we give an $\Omega(mn)$ lower bound on the message complexity to determine a feasible solution (suitable for both synchronous and asynchronous cases). In Section 4 we present an algorithm that finds a feasible solution to the problem whose message complexity is $O(mn \log m / \log n)$, which is then optimal when m is bounded by a polynomial in n .

Interestingly enough, message complexity is never affected by the value p , while running time is. We then show how to adapt the algorithm to work also in the asynchronous case at the expenses of a slight increase in message complexity; this time the message cost depends also on p , but the asymptotic bound is affected only when p is very large (*i.e.*, only if $p \notin O(m^m)$). In Section 5 we show that the proposed algorithm (both synchronous and asynchronous versions) computes a Balanced Coloring whose cost is only a factor of three off the optimal one, and we also show that the analysis of the approximation is tight. Finally, we show that we can find Balanced Colorings with a better approximation ratio at the expenses of the message and/or time complexity.

A preliminary version of this work appeared in [6]. In the previous version it was assumed that parameter p (the maximum number of items of a given color) was known to the computational agents. Since in practical situations it is difficult to have a good

estimate of such a global parameter, in this new version we removed this assumption. This required both new algorithmic ideas and technical efforts. The algorithm for the asynchronous communication model was also not contained in the preliminary version. Finally, we enriched the proof of the lower bound with new insights that could be useful for further generalizations to different network topologies.

2 Related problems and centralized version

In this section we relate the Distributed Balanced Color Assignment problem to known matching problems that have been well studied in centralized settings. We will first show that when $m = n$ our problem is equivalent to a maximum weight perfect matching problem on complete bipartite graphs. On the other hand, when $m \geq n$, our problem reduces to the weighted β -assignment problem.

The class of β -assignment problems has been introduced by Chang and Lee [4], in the context of the problems of assigning jobs to workers, in order to incorporate the problem of balancing the work load that is given to each worker. In the weighted β -assignment problem one aims at minimizing the maximum number of jobs assigned to each worker.

The interested reader can find useful references on these problems, their complexity, and related approximation issues in [1, 3, 15, 20, 22].

We associate to agents and colors the complete bipartite graph on $n + m$ vertices, which we denote by $G = (\mathcal{C}, \mathcal{A}, \mathcal{C} \times \mathcal{A})$. We add weights to G as follows: the weight of the edge joining agent a_i and color c_j is $Q_{j,i}$.

Case $m = n$. Given a graph (V, E) , a perfect matching is a subset M of edges in E such that no two edges in M share a common vertex and each vertex of V is incident to some edge in M . When edges of the graph have an associated weight, then a maximum weight perfect matching is a perfect matching such that the sum of the weights of the edges in the matching is maximum.

Lemma 3. *When $m = n$, a maximum weight perfect matching on G is a minimum cost solution to the balanced color assignment problem.*

Proof. Given a perfect matching $E \subseteq \mathcal{E} = \mathcal{C} \times \mathcal{A}$ on G , for every $(c_j, a_i) \in E$ we assign color c_j to agent a_i . As G is complete and E is a perfect matching on G , every color is assigned to one and only one agent and vice-versa. Moreover, the cost of any color assignment E can be written as $\sum_{e \in \mathcal{E} \setminus E} w(e)$, and this expression achieves its minimum when E is a maximum weight perfect matching. \square

Finding matchings in graphs is one of the most deeply investigated problems in Computer Science and Operations Research (see [17] for a comprehensive description of the different variants, theoretical properties, and corresponding algorithms). The best algorithm known to find a perfect matching in a bipartite graph is due to Hopcroft and Karp

[10], and runs in $O(|E|\sqrt{|V|})$ time, where V and E denote the vertex and edge sets, respectively. The best known algorithm for finding a maximum weight perfect matching is the *Hungarian method*, due to Kuhn [13], which runs in time $O(n^3)$.

Case $m \geq n$. The β -assignment problem is defined on a bipartite graph $G = (S, T, E)$ where (S, T) is the bipartition of the vertex set. A β -assignment of S in G is a subset of the edges $X \subseteq E$ such that, in the induced subgraph $G' = (S, T, X)$, the degree of every vertex in S is exactly one. Let $\beta(X)$ be the maximum degree, in G' , of vertices in T and let $\beta(G)$ be the minimum value of $\beta(X)$ among all possible β -assignments X . The weighted β -assignment problem consists of finding a β -assignment X with $\beta(X) = \beta(G)$ which maximizes the total weight of the edges in X . The following lemma is straightforward.

Lemma 4. *The balanced color assignment problem is a weighted β -assignment of \mathcal{C} in the complete bipartite graph $G = (\mathcal{C}, \mathcal{A}, \mathcal{C} \times \mathcal{A})$, with $\beta(G) = \lceil m/n \rceil$.*

The fastest known algorithm to solve the weighted β -assignment problem is due to Chang and Ho [3] and runs in $O(\max\{|S|^2|T|, |S||T|^2\})$ time, which in our case gives the bound $O(m^2n)$.

While the maximum weighted perfect matching problem (and its variants) has been widely investigated in the distributed setting (see [8, 9]), no distributed results are known for the weighted β -assignment problem.

A brute force approach. A brute force distributed solution to the problem can be obtained by asking all the agents to send their color information to one specific agent (a priori chosen or elected as the leader of the ring); such an agent will then solve the problem locally and send the solution back to all the other agents. The factor dominating the message complexity of the algorithm above is the information collecting stage. Indeed, each agent sends $O(m)$ non-basic messages, each corresponding to $O(\log p / \log n)$ basic messages, through $O(n)$ links, on the average. This results in a message complexity of $O(mn^2 \log p / \log n)$. On the other hand, we might think of an algorithm in which each agent selects the correct number of colors basing its choice just on local information (*e.g.* its label). This requires no communication at all, but, even if we are able to prove that the agents agree correctly on a balanced coloring, we have no guarantee on how good the solution is. As we already said, we show that we can do better than this.

3 Lower bound on message complexity

In this section we prove a lower bound on the message complexity of the problem that applies to both synchronous and asynchronous rings.

We prove the lower bound on a particular subset \mathcal{I} of the instances of the problem. Let n be even and let $m = (nt)/2$, for some integer t . Since we are only interested in asymptotic bounds, for the sake of simplicity, we will also assume that m is a multiple of n , i.e. $t/2 = m/n$ is an integer.

For any agent a_i , let $a_{i'}$ denote the agent at maximum distance from a_i on the ring. In the following we say that a color is assigned to the pair $(a_i, a_{i'})$, for $i = 0, \dots, n/2 - 1$, to mean that it is assigned to both agents of the pair. We also say that a set \mathcal{C} of colors is assigned to agent a iff all the colors in \mathcal{C} are assigned to a .

Let $\{\mathcal{C}_0, \dots, \mathcal{C}_{n/2-1}\}$ be a partition of the set of colors such that $|\mathcal{C}_j| = t$ for all $j = 0, \dots, n/2 - 1$. Set \mathcal{I} consists of all instances of the Distributed Balanced Color Assignment Problem such that for any $i = 0, \dots, n/2 - 1$, the following two conditions hold:

- (a) for any color $j \in \mathcal{C}_i = \{i_1, \dots, i_t\}$ both agents of pair $(a_i, a_{i'})$ hold at least one item of color j , i.e. $Q_{j,i} > 0, Q_{j,i'} > 0$;
- (b) neither a_i nor $a_{i'}$ hold colors not in \mathcal{C}_i .

Lemma 5. *Given an instance in \mathcal{I} , any optimal solution assigns to $(a_i, a_{i'})$ only colors from set \mathcal{C}_i , for $i = 0, \dots, n/2 - 1$.*

Proof. Consider any solution to an instance from set \mathcal{I} that assigns to the agent a_i a color h_0 initially held by some pair $(a_k, a_{k'})$, with $k \neq i$. Since any optimal solution is perfectly balanced on input instances of \mathcal{I} , there must be at least one color h_1 initially stored in $(a_i, a_{i'})$ that is assigned to some other agent, say a_p . The same argument can in turn be applied to a_p and so on until (since the number of colors/agents is finite) we fall back on a_k . Formally, there exists $0 \leq k \leq n/2 - 1$, $k \neq i$, such that $h_0 \in \mathcal{C}_k \neq \mathcal{C}_i$, and a sequence of indices k_0, k_1, \dots, k_l , with $k_0 = k$, $k_1 = i$ and $k_{l+1} = k$, such that

- color $h_0 \in \mathcal{C}_{k_0} (= \mathcal{C}_k)$ is assigned to agent $a_{k_1} (= a_i)$;
- color $h_1 \in \mathcal{C}_{k_1} (= \mathcal{C}_i)$ is assigned to agent a_{k_2} ;
- \vdots
- color $h_l \in \mathcal{C}_{k_l}$ is assigned to agent $a_{k_{l+1}} (= a_k)$.

Let $Cost_1$ denote the cost of such a solution and let Γ be the contribution to the cost given by colors different from h_0, h_1, \dots, h_l . Then, recalling condition (b) of the definition of \mathcal{I} , we have

$$\begin{aligned} Cost_1 &= \Gamma + \sum_{\substack{w=0, \\ w \neq k_1}}^{n-1} Q_{h_0, w} + \sum_{\substack{w=0, \\ w \neq k_2}}^{n-1} Q_{h_1, w} + \dots + \sum_{\substack{w=0, \\ w \neq k_{l+1}}}^{n-1} Q_{h_l, w} \\ &= \Gamma + (Q_{h_0, k_0} + Q_{h_0, k'_0}) + \dots + (Q_{h_l, k_l} + Q_{h_l, k'_l}). \end{aligned}$$

Consider now a solution that differs from the previous one only by the fact that every color in \mathcal{C}_w is assigned to agent a_w for $w = k_0, k_1, \dots, k_l$. Namely,

- $h_0 \in \mathcal{C}_{k_0}$ is assigned to a_{k_0} ;
- $h_1 \in \mathcal{C}_{k_1}$ is assigned to a_{k_1} ;
- \vdots

- $h_l \in \mathcal{C}_{k_l}$ is assigned to a_{k_l} .

This is clearly a perfectly balanced solution, since each agent “loses” and “gains” exactly one color with respect to the previous case. Letting $Cost_2$ be the cost of such a solution, we have

$$\begin{aligned} Cost_2 &= \Gamma + \sum_{\substack{w=0, \\ w \neq k_0}}^{n-1} Q_{h_0,w} + \cdots + \sum_{\substack{w=0, \\ w \neq k_l}}^{n-1} Q_{h_l,w} \\ &= \Gamma + Q_{h_0,k'_0} + \cdots + Q_{h_l,k'_l}. \end{aligned}$$

Hence,

$$Cost_1 - Cost_2 = Q_{h_0,k_0} + \cdots + Q_{h_l,k_l} > 0,$$

where the inequality follows from condition (1) of the definition of \mathcal{I} . \square

We now consider two specific instances in \mathcal{I} that will be used in the following proofs.

For each pair $(a_i, a_{i'})$, for $i = 0, \dots, n/2 - 1$, and its initially allocated set of colors $\mathcal{C}_i = \{i_1, \dots, i_t\}$, fix any $u > 1$ and partition set \mathcal{C}_i into subsets \mathcal{C}' and \mathcal{C}'' , each of cardinality $t/2$. We define instance $\mathcal{I}_1 \in \mathcal{I}$ for the pair $(a_i, a_{i'})$ in the following way:

$$\begin{aligned} \mathcal{I}_1 : \quad & Q_{j,i} = u && \text{for each } j \in \mathcal{C}_i \\ & Q_{j,i'} = Q_{j,i} = u && \text{for each } j \in \mathcal{C}' \\ & Q_{j,i'} = Q_{j,i} + 1 = u + 1 && \text{for each } j \in \mathcal{C}'' \end{aligned}$$

Hence, by construction, instance \mathcal{I}_1 has the property that for any $j \in \mathcal{C}''$, $Q_{j,i'} > Q_{j,i}$.

Example 6. Consider a pair $(a_i, a_{i'})$ with a set of colors $\mathcal{C}_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Let $u = 2$. If $\mathcal{C}' = \{2, 4, 5, 8\}$ and $\mathcal{C}'' = \{1, 3, 6, 7\}$, then instance \mathcal{I}_1 will be as follows:

colors	1	2	3	4	5	6	7	8
# items for a_i	2	2	2	2	2	2	2	2
# items for $a_{i'}$	3	2	3	2	2	3	3	2

In the following lemma we will show that the only optimal solution to \mathcal{I}_1 is the one that assigns \mathcal{C}' to a_i and \mathcal{C}'' to $a_{i'}$. The above example gives an intuition of the formal proof. By Lemma 5, we know that only the items that need to be exchanged between a_i and $a_{i'}$ account for the cost of the optimal solution, and the latter is achieved by moving items with weight 2 (those highlighted in bold in the table), i.e., by assigning \mathcal{C}' to a_i and \mathcal{C}'' to $a_{i'}$, for a total cost of 16.

Lemma 7. *The only optimal solution to instance \mathcal{I}_1 is the one that assigns \mathcal{C}' to a_i and \mathcal{C}'' to $a_{i'}$.*

Proof. We first compute the cost of this solution:

$$Cost = \sum_{j \in \mathcal{C}'} Q_{j,i'} + \sum_{j \in \mathcal{C}''} Q_{j,i} = \sum_{j \in \mathcal{C}'} Q_{j,i} + \sum_{j \in \mathcal{C}''} Q_{j,i} = \sum_{j \in \mathcal{C}_i} Q_{j,i}.$$

Consider any other partition of \mathcal{C}_i into two sets $\overline{\mathcal{C}'}$ and $\overline{\mathcal{C}''}$. Consider another solution that assigns $\overline{\mathcal{C}'}$ to a_i and $\overline{\mathcal{C}''}$ to $a_{i'}$ and let us compute the cost of this new solution:

$$\begin{aligned} \overline{Cost} &= \sum_{j \in \overline{\mathcal{C}'}} Q_{j,i'} + \sum_{j \in \overline{\mathcal{C}''}} Q_{j,i} \\ &= \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}'} Q_{j,i'} + \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}''} Q_{j,i'} + \sum_{j \in \overline{\mathcal{C}''}} Q_{j,i} \\ &= \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}'} Q_{j,i} + \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}''} Q_{j,i'} + \sum_{j \in \overline{\mathcal{C}''}} Q_{j,i} \\ &= \sum_{j \in \mathcal{C}_i \setminus (\overline{\mathcal{C}'} \cap \mathcal{C}'')} Q_{j,i} + \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}''} Q_{j,i'} \\ &> \sum_{j \in \mathcal{C}_i \setminus (\overline{\mathcal{C}'} \cap \mathcal{C}'')} Q_{j,i} + \sum_{j \in \overline{\mathcal{C}'} \cap \mathcal{C}''} Q_{j,i} = Cost, \end{aligned}$$

where the inequality follows by observing that

- there is at least one $j \in \overline{\mathcal{C}'} \cap \mathcal{C}''$, otherwise the two partitions would coincide;
- on instance \mathcal{I}_1 we have that for every $j \in \mathcal{C}''$, $Q_{j,i'} > Q_{j,i}$.

□

We now define the instance $\mathcal{I}_2 \in \mathcal{I}$ for the pair $(a_i, a_{i'})$ in the following way:

$$\begin{aligned} \mathcal{I}_2 : \quad Q_{j,i'} &= u && \text{for each } j \in \mathcal{C}_i \\ Q_{j,i} &= Q_{j,i'} = u && \text{for each } j \in \mathcal{C}' \\ Q_{j,i} &= Q_{j,i'} - 1 = u - 1 && \text{for each } j \in \mathcal{C}'' \end{aligned}$$

where $\mathcal{C}_i, \mathcal{C}', \mathcal{C}''$, and u are set as before. By construction, instance \mathcal{I}_2 has now the property that for any $j \in \mathcal{C}''$, $Q_{j,i'} < Q_{j,i}$.

Example 8. Consider again the pair $(a_i, a_{i'})$ on the same set of colors \mathcal{C}_i and same partition $\mathcal{C}' = \{2, 4, 5, 8\}$, $\mathcal{C}'' = \{1, 3, 6, 7\}$, and same $u = 2$, exactly as in Example 6. Instance \mathcal{I}_2 will be as follows (the cost of the optimal solution is equal to 12 and highlighted in bold):

colors	1	2	3	4	5	6	7	8
# items for a_i	2	2	2	2	2	2	2	2
# items for $a_{i'}$	1	2	1	2	2	1	1	2

Observe that, from a_i point of view, instances \mathcal{I}_1 and \mathcal{I}_2 are indistinguishable. Nevertheless, the optimal solution for instance \mathcal{I}_2 is to assign to a_i the complement set of indices with respect to the optimal solution to instance \mathcal{I}_1 .

Analogously as the previous lemma we can prove the following result.

Lemma 9. *There is only one optimal solution for instance \mathcal{I}_2 : assign colors in \mathcal{C}' to $a_{i'}$ and colors in \mathcal{C}'' to a_i .*

Proof. The proof is very similar to that of Lemma 7. The cost of the solution defined in the statement is now:

$$Cost = \sum_{j \in \mathcal{C}'} Q_{j,i} + \sum_{j \in \mathcal{C}''} Q_{j,i'} = \sum_{j \in \mathcal{C}'} Q_{j,i'} + \sum_{j \in \mathcal{C}''} Q_{j,i} = \sum_{j \in \mathcal{C}_i} Q_{j,i'}.$$

The cost of any other solution is calculated as in the proof of Lemma 7 with the exception that now instance \mathcal{I}_2 has the property that for any $j \in \mathcal{C}''$, $Q_{j,i'} < Q_{j,i}$. \square

The core of the lower bound's proof lies in the simple observation that agent a_i is not able to distinguish between instance \mathcal{I}_1 and instance \mathcal{I}_2 without knowing also the quantities $Q_{j,i'}$ for colors j falling into partition \mathcal{C}'' .

Lemma 10. *If agent a_i knows at most $t/2$ colors held by $a_{i'}$, it cannot compute its optimal assignment of colors.*

Proof. Construct a partition of \mathcal{C}_i in the following way: place index j in \mathcal{C}' if a_i has knowledge of $Q_{j,i'}$ and in \mathcal{C}'' in the other case. If the cardinality of \mathcal{C}' is smaller than $t/2$, arbitrarily add indices to reach cardinality $t/2$. Agent a_i cannot distinguish between instances \mathcal{I}_1 and \mathcal{I}_2 constructed according to this partition of \mathcal{C}_i and, hence, by lemmas 7 and 9 cannot decide whether it is better to keep colors whose indices are in \mathcal{C}' or in \mathcal{C}'' . Finally, observe that in both instances indices in \mathcal{C}' are exactly in the same position in the ordering of the colors held by $a_{i'}$, thus the knowledge of these positions does not help. \square

Theorem 11. *The message complexity of the distributed color assignment problem on ring is $\Omega(mn)$.*

Proof. Let **A** be any distributed algorithm for the problem running on instances in \mathcal{I} . By the end of the execution of **A**, each agent has to determine its own assignment of colors. Fix any pair $(a_i, a_{i'})$ and consider the time at which agent a_i decides its own final assignment of colors. Assume that at this time a_i knows information about at most $t/2 = m/n$ colors of agent $a_{i'}$. By Lemma 10, it cannot determine an assignment of colors for itself yielding the optimal solution.

Therefore, for all $n/2$ pairs $(a_i, a_{i'})$, agent a_i has to get information concerning at least m/n of the colors held by $a_{i'}$. We use Shannon's Entropy to compute the minimum number of bits B to be exchanged between any pair $(a_i, a_{i'})$ so that this amount of information is known by a_i . We have:

$$B = \log \binom{m}{\frac{m}{n}}.$$

Using Stirling's approximation and the inequality $m \geq n$, we get

$$\begin{aligned} B &\approx \frac{m}{n} \cdot \log \frac{m \cdot n}{m + n} \\ &\geq \frac{m}{n} \cdot \log \frac{n}{2} \in \Omega\left(\frac{m}{n} \log n\right). \end{aligned}$$

As a basic message contains $\log n$ bits, any pair $(a_i, a_{i'})$ needs to exchange at least $\Omega(m/n)$ basic messages. Each such message must traverse $n/2$ links of the ring to get to one agent of the pair to the other. As we have $n/2$ pairs of agents, the lower bound on message complexity is given by

$$\Omega(m/n) \cdot \frac{n}{2} \cdot \frac{n}{2} \in \Omega(m \cdot n).$$

□

4 A distributed message-optimal algorithm

In this section we first describe an algorithm that exhibits optimal message complexity on synchronous ring. We will then show how to adapt the algorithm to the case of an asynchronous ring. In the next section we will prove that the algorithm is guaranteed to compute an approximation of the color assignment that is within a factor three from the optimal solution (for both synchronous and asynchronous ring).

4.1 Synchronous ring

At a high level, the algorithm consists of three phases: in the first phase, the algorithm elects a leader a_0 among the set of agents. The second phase of the algorithm is devoted to estimate the parameter $p = \max_i \max_j Q_{j,i}$, *i.e.* the maximum number of items of a given color held by agents. Finally, the last phase performs the assignment of colors to agents in such a way to be consistent with Definition 1. In the following we describe the three phases in detail.

Algorithm Sync-Balance

Phase 1. The first phase is dedicated to leader election that can be done in $O(n)$ time with a message complexity of $O(n \log n)$ on a ring of n nodes, even when the nodes are not aware of the size n of the ring [11].

Leader election has also been studied in arbitrary wired networks [7]. An $O(n \text{ polylog}(n))$ time deterministic algorithm is available even for ad hoc radio networks of *unknown and arbitrary* topology without a collision detection mechanism, even though

the size of the network must be known to the algorithm code (see [5] for the currently best result).

Without loss of generality, in the following we will assume that agent a_0 is the leader and that a_1, a_2, \dots, a_{n-1} are the other agents visiting the ring clockwise. In the rest of this paper, we will refer to agent $a_{i-1 \bmod n}$ (resp. $a_{i+1 \bmod n}$) as to the *preceding* (resp. *following*) neighbor of a_i .

Phase 2. In this phase agents agree on an upper bound p' of p such that $p' \leq 2p$.

Given any agent a_i and an integer $r \geq 0$, we define:

$$B_i(r) = \begin{cases} 1 & \text{if } \max_j Q_{j,i} = 0 \text{ and } r = 0; \\ 1 & \text{if } 2^r \leq \max_j Q_{j,i} < 2^{r+1} \text{ and } r > 0; \\ 0 & \text{otherwise.} \end{cases}$$

This phase is organized in consecutive stages labeled $0, 1, \dots$. At stage $r = 0$, the leader sets an integer variable A to zero, which will be updated at the end of each stage and used to determine when to end this phase.

In stage $r \geq 0$, agent a_i , for $i = 0, 1, \dots, n-1$, waits for i time units from the beginning of the stage. At that time a message M might arrive from its preceding neighbor. If no message arrives, then it is assumed that $M = 0$. Agent a_i computes $M = M + B_i(r)$ and, at time unit $i + 1$, sends M to its following neighbor only if $M > 0$, otherwise it remains silent.

After n time slots in stage r , if the leader receives a message $M \leq n$ from the preceding neighbor, then it updates variable $A = A + M$, and, if $A < n$, proceeds to stage $r + 1$ of Phase 2; otherwise it sends a message clockwise on the ring containing the index of the last stage ℓ performed in Phase 2. Each agent then computes $p' = 2^{\ell+1}$, forwards the message clockwise, waits for $n - i + 1$ time units and then proceeds to Phase 3.

Lemma 12. *Phase 2 of Algorithm Sync-Balance computes an upper bound p' of p such that $p' \leq 2p$ within $O(n \log p)$ time units and using $O(n^2)$ basic messages.*

Proof. We will say that agent a_i *speaks up* in stage r when $B_i(r) = 1$. Throughout the execution of the algorithm, integer variable A records the number of agents that have spoken up so far.

Any agent a_i speaks up in one stage only. Indeed, given the color j' for which agent a_i has the maximum number of items, then $B_i(r) = 1$ only at stage r such that $Q_{j',i}$ falls in the (unique) interval $[2^r, 2^{r+1})$. Let a_{i^*} be the agent having the largest amount of items of the same color among all agents, *i.e.*, such that $Q_{j^*,i^*} = p$, for some $j^* \in [0, m-1]$. Then $B_{i^*}(r) = 1$ for stage r such that $2^r \leq p < 2^{r+1}$, *i.e.*, agent a_{i^*} speaks up when $r = \ell$. Observe that at the end of stage ℓ the leader sets $A = n$, as all n agents must have spoken up by that time. Therefore, considering also the last extra stage in which the agents are informed of the value of ℓ , Phase 2 ends after $\ell + 2$ stages, *i.e.* $n(2 + \log p)$ time units.

For what concerns message complexity, in each stage, for $r = 0, \dots, \log p$, either no messages are sent, or a message traverses a portion of the ring. Observe that, as each agent speaks up only once during this phase, messages circulating on the ring must always be

originated by different agents. Hence, the number of stages in which a message circulates on the ring is at most n and there must be at least $\max\{0, \log p - n\}$ silent stages. In conclusion, Phase 2 message complexity is bounded by $O(n^2)$.

As for the ratio between the actual value of p and its approximation p' computed in Phase 2, by construction we have that $2^\ell \leq p$ and

$$p' = 2^{\ell+1} = 2 \cdot 2^\ell \leq 2p.$$

□

Phase 3. As a preliminary step, each agent a_i computes the number of colors it will assign to itself and stores it in a variable \mathcal{K}_i . Namely, each agent a_i , for $i = 0, 1, \dots, n-1$ computes $g = (\lfloor \frac{m}{n} \rfloor + 1) n - m$ and then sets \mathcal{K}_i as follows (recall Definition 1):

$$\mathcal{K}_i = \begin{cases} \lfloor \frac{m}{n} \rfloor & \text{if } i < g; \\ \lfloor \frac{m}{n} \rfloor + 1 & \text{otherwise.} \end{cases} \quad (2)$$

In the rest of this phase, the agents agree on a color assignment such that each agent a_i has exactly \mathcal{K}_i colors. Algorithms 1 and 2 report the pseudo-code of the protocol performed by a general agent a_i in this phase and that is here described.

Let p' be the upper bound on p computed in Phase 2. Phase 3 consists of $\log p' + 1$ stages. In each stage r , for $r = 0, \dots, \log p'$, the agents take into consideration only colors whose weights fall in interval $I_r = [l_r, u_r)$ defined as follows:

$$\begin{cases} I_0 &= \left[\frac{p'}{2}, +\infty \right), \\ I_r &= \left[\frac{p'}{2^{r+1}}, \frac{p'}{2^r} \right) \text{ for } 0 < r < \log p' \\ I_{\log p'} &= [0, 1) \end{cases} \quad (3)$$

Observe that in consecutive stages, agents consider weights in decreasing order, as $u_{r+1} \leq l_r$.

At the beginning of each stage r , all agents have complete knowledge of the set of colors \mathcal{C}_{r-1} that have already been assigned to some agent in previous stages. At the beginning of this phase, \mathcal{C}_{-1} is the empty set, and after the last stage is performed, $\mathcal{C}_{\log p'}$ must be the set of all colors.

Stage r is, in general, composed of two steps; however, the second step might not be performed, depending on the outcome of the first one. In the first step, the agents determine if there is at least one agent with a weight falling in interval I_r , by forwarding a message around the ring only if one of the agents is in this situation. If a message circulates on the ring in step one, then all agents proceed to step two in order to assign colors whose weight fall in interval I_r and to update the set of assigned colors. Otherwise, step two is skipped. Now, if there are still colors to be assigned (*i.e.*, if $\mathcal{C}_r \neq \mathcal{C}$), all agents proceed to stage $(r+1)$; otherwise, the algorithm ends. In more details:

STEP 1. Agent a_i (leader included) waits i time units (zero for the leader) from the beginning of the stage, and then acts according to the following protocol:

Case 1: If a_i receives a message from its preceding neighbor containing the label k of some agent a_k , it simply forwards the same message to its following neighbor and waits for $(n + k - i - 1)$ time units;

otherwise

Case 2: If a_i has a weight falling into interval I_r , then it sends a message containing its label i to its following neighbor and waits for $(n - 1)$ time units;

otherwise

Case 3: It does nothing and waits for n time units.

If Case 1 or Case 2 occurred, then agent a_i knows that STEP 2 is to be performed and that it is going to start after waiting the designed time units.

Otherwise, if Case 3 occurred, after n units of time, agent a_i might receive a message (containing label k) from its preceding neighbor, or not. If it does, then a_i learns that Case 2 occurred at some agent a_k having label $k > i$ and that STEP 2 is to be performed. Hence, it forwards the message to its following neighbor in order to inform all agents having labels in the interval $[i + 1, \dots, k - 1]$, unless this interval is empty (meaning that a_i was the last agent to be informed). Then, after waiting for another $(k - i - 1)$ time units, agent a_i proceeds to STEP 2. On the contrary, if a_i got no message, it learns that Case 2 did not occur at any agent and hence, STEP 2 needs not be performed. After waiting for $(n - i)$ time units, a_i can proceed to the next stage $(r + 1)$.

Observe that, when STEP 2 has to be performed, STEP 1 lasts exactly $n + k - 1 < 2n$ time units for all agents, where k is the smallest agent's label at which Case 2 occurs, while it lasts exactly $2n$ time units for all agents in the opposite case. Indeed, referring to the pseudo-code in Algorithm 1, completion time is given by the sum of the time units in the following code lines: in Case 1 of lines 7 and 10 ($i \neq k$); in Case 2 of lines 7 and 15 ($i = k$); in Case 3 of lines 7, 18 and 22 if agents proceed to STEP 2 ($i \neq k$), and lines 7 and 26 otherwise.

As the time needed by agents to agree on skipping STEP 2 is larger than the time needed to agree in performing it, it is not possible that some agent proceeds to STEP 2 and some other to stage $(r + 1)$. On the contrary, agents are perfectly synchronized to proceed to STEP 2 or stage $(r + 1)$.

STEP 2. When this step is performed, there exists a non empty subset of agents having at least one weight falling into interval I_r . Only these agents actively participate to the color assignment phase, while the others just forward messages and update their list of assigned colors. Color assignment is done using a greedy strategy: agent a_i assigns itself the colors it holds which fall into interval I_r and that have not been already assigned to other agents. Once a color is assigned to an agent, it will never be re-assigned to another one.

To agree on the assignment, the agents proceed in the following way: agent a_i creates the list $\mathcal{L}_{i,r}$ of colors it holds whose weights fall into interval I_r and that have not been

assigned in previous stages. Then, a_i waits i time units (zero for the leader) from the beginning of the step. At that time, either a_i receives a message \mathcal{M} from its preceding neighbor or not. In the first case, the message contains the set of colors assigned in this stage to agents closer to the leader (obviously, this case can never happen to the leader). Agent a_i then checks if there are some colors in its list $\mathcal{L}_{i,r}$ that are not contained in \mathcal{M} (empty message in the case of the leader), and then assigns itself as many such colors as possible, without violating the constraint \mathcal{K}_i on the maximum number of colors a single agent might be assigned. Then, a_i updates message \mathcal{M} by adding the colors it assigned itself, and finally sends the message to its following neighbor. If $\mathcal{L}_{i,r}$ is empty, or it contains only already assigned colors, a_i just forwards message \mathcal{M} as it was. In both cases, a_i then waits for a new message \mathcal{M}' that will contain the complete list of colors assigned in this stage. \mathcal{M}' is used by all a_i to update the list of already assigned colors and is forwarded on the ring. When the message is back to the leader, stage $(r + 1)$ can start.

Lemma 13. *Let K_r be the number of colors assigned in stage r of Phase 3, then stage r can be completed in at most $O(n)$ time units using at most $O\left(n \cdot \frac{K_r \log m}{\log n}\right)$ basic messages.*

Proof. The bound on the time complexity follows straightforwardly by observing that each of the two steps requires at most $2n$ time units.

For what concerns message complexity, STEP 1 requires no messages if STEP 2 is skipped, and $n - 1$ otherwise. In fact, only one basic message goes clockwise on the ring from a_k to a_{k-1} , where k is the smallest index at which Case 2 occurs. The worst case for STEP 2 is the case in which the leader itself assigns some colors, as a possibly long message containing color ID's must go twice around the ring. As there are m colors, one color can be codified using $\log m$ bits, then, sending K_r colors requires no more than $\frac{K_r \log m}{\log n}$ basic messages. In conclusion, the total number of basic messages is upper bounded by $O\left(n \cdot \frac{K_r \log m}{\log n}\right)$. \square

Corollary 14. *Phase 3 of Algorithm Sync-Balance can be completed within $O(n \log p)$ time units and using $O(nm \cdot \frac{\log m}{\log n})$ basic messages.*

Proof. It will suffice to sum up the worst cases for message and time complexity from Lemma 13 over all stages $r = 0, \dots, \log p'$, where $p' \leq 2p$ (Lemma 12).

The upper bound on the time complexity is straightforward. Let K_r be defined as in the statement of Lemma 13, i.e. as the number of colors assigned in a generic stage r of Phase 3. The upper bound on the message complexity follows by observing that $\sum_{r=0}^{\log p'} K_r = m$, as the total number of assigned colors during the $\log p' + 1$ stages is exactly the given number of colors. \square

We are now ready to prove that our algorithm is correct. In Section 5 we will evaluate the ratio of the cost of the solution found by this algorithm and the one of the optimal solution.

Algorithm 1 Sync-Balance - Phase 3 (performed by agent a_i)

Require: p' computed in Phase 2 \triangleright upper bound to maximum number of items of the same color

- 1: Compute \mathcal{K}_i \triangleright Number of colors a_i has to be assigned, as defined in Equation (2)
- 2: $\mathcal{C}_{-1} \leftarrow \emptyset$ \triangleright set of colors assigned up to the previous stage
- 3: **for** $r = 0$ **to** $\log p'$ **do**
- 4: $\mathcal{L}_{i,r} = \{c_j \mid c_j \notin \mathcal{C}_{r-1} \text{ and } Q_{j,i} \in I_r\}$ \triangleright Colors assignable to a_i in stage r . Intervals I_r are defined in (3)
- \triangleright Begin of STEP 1
- 5: Wait i time units
- 6: **if** Got message $\mathcal{M} = \{k\}$ from its preceding neighbor **then** \triangleright Case 1
- $\triangleright k < i$ is an agent label
- 7: Forward message \mathcal{M} to its following neighbor
- 8: Wait $n - i + k - 1$ time units
- 9: Step 2 \triangleright proceeds to STEP 2
- 10: **else**
- 11: **if** $\mathcal{L}_{i,r} \neq \emptyset$ **then** \triangleright Case 2
- 12: Send message $\mathcal{M} = \{i\}$ to its following neighbor
- 13: Wait $n - 1$ time units
- 14: Step 2 \triangleright proceeds to STEP 2
- 15: **else** \triangleright Case 3
- 16: Wait n time units
- 17: **if** Got message $\mathcal{M} = \{k\}$ from its preceding neighbor **then**
- 18: **if** $k - i - 1 > 0$ **then** \triangleright informs other agents that Step 2 is to be performed
- 19: Forward message \mathcal{M} to its following neighbor
- 20: Wait for $k - i - 1$
- 21: Step 2 \triangleright procedure call to STEP 2
- 22: **end if**
- 23: **else**
- 24: Wait $n - i$ time units \triangleright proceeds to next stage skipping STEP 2
- 25: **end if**
- 26: **end if**
- 27: **end if**
- 28: **end for**

Algorithm 2 Sync-Balance - Phase 3 STEP 2 (performed by agent a_i)

```
1: procedure STEP 2
2:   Wait  $i$  time units
3:   if Got message  $\mathcal{M}$  from preceding neighbor with list of colors then
4:      $\mathcal{L}_{i,r} \leftarrow \mathcal{L}_{i,r} \setminus \mathcal{M}$  ▷ list of candidate colors to self assign
5:   else
6:     Create empty message  $\mathcal{M}$ 
7:   end if
8:   if  $|\mathcal{L}_{i,r}| \neq \emptyset$  then
9:     Self assign maximum number of colors among those in  $\mathcal{L}_{i,r}$ 
▷ the total number of colors  $a_i$  can assign itself is given by  $\mathcal{K}_i$ 
10:    Add self assigned colors to  $\mathcal{M}$ 
11:  end if
12:  if  $\mathcal{M} \neq \emptyset$  then
13:    Send message  $\mathcal{M}$  to the following neighbor
14:  end if
15:  Wait for message  $\mathcal{M}'$  from preceding neighbor with list of colors
16:   $\mathcal{C}_r \leftarrow \mathcal{C}_{r-1} \cup \mathcal{M}'$  ▷ updates set of assigned colors
17:  Forward message  $\mathcal{M}'$  to the following neighbor
18:  if  $\mathcal{C}_r = \mathcal{C}$  then ▷ all colors have been assigned
19:    stop
20:  else
21:    Wait for  $n - i$  time units
22:  end if
23: end procedure
```

Theorem 15. *Assuming $m \in O(n^c)$, for some constant c , Algorithm **Sync-Balance** finds a feasible solution to the balanced color assignment problem in time $O(n \log p)$ using $\Theta(mn)$ messages.*

Proof. To prove correctness, we show that any assignment of colors to agents computed by algorithm **Sync-Balance** satisfies the two following conditions:

(i) A color c_j cannot be assigned to more than one agent.

(ii) All colors are assigned.

(i) The algorithm can assign a new color c_j to agent a_i only in line 9 of Algorithm 2. This can only happen if c_j has not been already assigned in a previous stage, or in the current stage to an agent with smaller label. Since, in the stage, the color assignment is done sequentially (starting from the leader and following the ring clockwise), no color can be assigned to two different agents. Moreover, in lines 15-17 of Algorithm 2, all agents update the list of colors assigned in the current stage and, hence, in later stages, already assigned colors will not be assigned again. Therefore **Sync-Balance** prevents the assignment of the same color to two different agents.

(ii) If an available color c_j of weight $Q_{j,i} \in [l_r, u_r)$ is not taken by a_i during stage r , it is only because a_i has enough colors already (line 9). However, this circumstance may not occur at all agents during the same stage (for this would imply that there were more than m colors). Thus, either the color is taken by a higher labeled agent in stage r , or is “left free” for agents for which the weight of c_j is less than l_r . By iterating the reasoning, we may conclude that, if not taken before, the color must be eventually assigned in stage $\lceil \log p \rceil + 1$, where agents are allowed to pick colors for which their weight is zero.

As for upper bounds on time and message complexities, by summing up upper bounds for the three phases, we have

$$\text{Time complexity: } \underbrace{O(n)}_{\text{Phase 1}} + \underbrace{O(n \log p)}_{\text{Phase 2}} + \underbrace{O(n \log p)}_{\text{Phase 3}} = O(n \log p),$$

$$\text{Message complexity: } \underbrace{O(n \log n)}_{\text{Phase 1}} + \underbrace{O(n^2)}_{\text{Phase 2}} + \underbrace{O\left(nm \cdot \frac{\log m}{\log n}\right)}_{\text{Phase 3}} = O(nm),$$

where we used Lemma 12, Corollary 14, and the facts that $m \geq n$ and that $\log m / \log n \in O(1)$, under the given hypothesis. □

4.2 Asynchronous ring

In an asynchronous ring such instructions as “wait for i time units” (see Algorithm 1 and 2) cannot guarantee a correct completion of the global algorithm. Here we discuss

how to make simple modifications to **Sync-Balance** in order to get an algorithm (named **Async-Balance**) that correctly works in the asynchronous case as well.

The leader election in Phase 1 can be done in $O(n)$ time with a message complexity of $O(n \log n)$ even on an asynchronous ring of n nodes [11]. Therefore, the main differences are in Phase 2 and Phase 3.

In Phase 2 we propose a slightly different strategy that works in only 2 stages, instead of $\log p$. This better time complexity translates, in general, into an extra cost in terms of message complexity. Nevertheless, under reasonable hypothesis (namely when $p \in O(m^m)$), the message complexity reduces to the same bound as for the synchronous setting.

Finally, in Phase 3, the main ideas remain the same, but there are no “silent stages” and the leader acts differently from the other agents, as it is the one originating all messages circulating on the ring.

In the following we highlight the main differences with the synchronous protocol:

Algorithm Async-Balance

Phase 1. Leader election can be accomplished with an $O(n \log n)$ message complexity [11].

Phase 2. This phase consists of only two stages. In the first stage the agents compute $p = \max_i \max_j Q_{j,i}$. Let $p_i = \max_j Q_{j,i}$, *i.e.* the maximum number of items of the same color agent a_i possesses. The leader originates a message containing p_0 . Upon reception of a message M from its preceding neighbor, agent a_i computes $M = \max\{M, p_i\}$ and forwards M to its following neighbor. The message that gets back to the leader contains p and it is forwarded once again on the ring to inform all agents.

Observe that Phase 2 requires no more than $O\left(n \cdot \frac{\log p}{\log n}\right)$ basic messages, as $O(\log p / \log n)$ basic messages are needed to send the p_i 's and p .

Phase 3. Changes in this phase concern both the execution of STEP 1 and STEP 2, that are to be modified in the following way:

STEP 1. Each agent a_i computes its list of assignable colors $\mathcal{L}_{i,r}$ and sets $Y_i(r) = 1$ if $|\mathcal{L}_{i,r}| > 0$, and $Y_i(r) = 0$ otherwise. The leader starts the step by sending, to its following neighbor, a basic boolean message containing $Y_0(r)$. Upon reception of a message M from its preceding neighbor, agent a_i computes $M = M \vee Y_i(r)$ and forwards M to its following neighbor. When the leader gets the message back, it forwards the message again on the ring, and the same is done by all agents, until the message arrives to a_{n-1} . The second time one agent (leader included) gets the message, it checks its content: if it is a one, then it knows that it has to proceed to STEP 2; otherwise, if it contains a zero, it proceeds to the next stage.

STEP 2. The leader starts the step by sending, to its following neighbor, a (possibly empty) list of self assigned colors, obtained exactly as in the synchronous case. Then agents act as in the synchronous protocol, with the exception that they are activated by the arrival of a message from the preceding neighbor and not by a time stamp. Agents

proceed to the next stage after forwarding the complete list of colors assigned in the stage.

Lemma 16. *Let K_r be the number of colors assigned in stage r of Phase 3, then stage r can be completed using at most $O\left(n \cdot \frac{K_r \log m}{\log n}\right)$ basic messages.*

Proof. STEP 1 is always performed and a basic message is forwarded (almost²) twice around the ring. Hence, $O(n)$ basic messages are used. When STEP 2 is performed, a message containing color ID's goes (almost) twice around the ring. Analogously to the synchronous case, we can prove that no more than $O\left(n \cdot \frac{K_r \log m}{\log n}\right)$ messages are needed. \square

Analogously to the synchronous case, we can prove the following corollary.

Corollary 17. *Phase 3 of Algorithm Async-Balance can be completed using $O\left(nm \cdot \frac{\log m}{\log n}\right)$ basic messages.*

Theorem 18. *Assuming $m \in O(n^c)$, for some constant c , Algorithm Async-Balance finds a feasible solution to the balanced color assignment problem, on asynchronous rings, within time $O(n \log p)$ using $O\left(n \cdot \frac{\log p}{\log n} + nm\right)$ basic messages.*

Proof. The correctness proof is analogous to the synchronous case.

The time complexity is asymptotically equivalent to the synchronous case. Indeed, as already mentioned, the leader election in Phase 1 can be completed in $O(n)$ time, Phase 2 requires 2 circles around the ring and, finally, Phase 3 includes $O(\log p)$ stages, each of them requiring 2 circles around the ring.

For what concerns message complexity, summing up upper bounds for single phases, we get

$$O(n \log n) + O\left(n \cdot \frac{\log p}{\log n}\right) + O\left(nm \cdot \frac{\log m}{\log n}\right) = O\left(n \cdot \frac{\log p}{\log n} + nm\right),$$

as $m \in O(n^c)$. \square

When we also have that $O(\log p) = O(m \log m)$, the algorithm exhibits the same *optimal* message complexity as in the synchronous setting. Namely, we can state the following result.

Corollary 19. *If $m \in O(n^c)$, for some constant c , and $p \in O(m^m)$, then Algorithm Async-Balance finds a feasible solution to the balanced color assignment problem, on asynchronous rings, using $\Theta(nm)$ messages.*

²On the second stage, agent a_{n-1} stops the message.

5 Approximation Factor of Algorithm Balance

The main result of this section is that the cost of the solution (as defined in Definition 2) computed by the algorithms presented in the previous sections is at most a small *constant* factor larger than the cost of the optimal solution. Namely, we will show that it is at most three times the optimal solution and that the analysis is tight. Moreover, we will show how to modify the algorithm to get a 2-approximation ratio at the expenses of a little increase of message complexity, and, for the synchronous case only, how to get a $(2 + \epsilon)$ -approximation ratio (for every $0 < \epsilon < 1$) at the expenses of an increase in time complexity.

Since, under the same assumptions of Corollary 19, the cost of the solution is the same both in the synchronous and asynchronous versions (the assignment of colors is exactly the same in both cases), in this section we will address both **Sync-Balance** and **Async-Balance** with the generic name **Balance**. In the following some results are expressed in terms of the value p' (respectively, p) computed by the agents in the synchronous (resp. asynchronous) case during Phase 2 of the algorithm. As these results hold for both p' and p , to avoid repeating the distinction between p' and p over and over again, we will indicate with \hat{p} both values p' and p .

We begin with the following lemma:

Lemma 20. *Let color c_j be assigned to agent a_i in stage r (of Phase 3) by algorithm **Balance**. Let a_k be a different agent such that $Q_{j,k} \in [l_r, u_r)$. Then $Q_{j,i} \leq 2 \cdot Q_{j,k}$.*

Proof. If $r = \lceil \log \hat{p} \rceil + 1$ (i.e., is the last stage), then it must be $Q_{j,i} = Q_{j,k} = 0$, and we are done. Otherwise, as c_j is assigned to agent a_i in stage r then it must be $Q_{j,i} \in [l_r, u_r)$ and the thesis easily follows from

$$\frac{\hat{p}}{2^{r+1}} \leq Q_{j,i}, Q_{j,k} < \frac{\hat{p}}{2^r}.$$

□

Let $B : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ be the assignment of colors to agents determined by algorithm **Balance**, and let $OPT : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ be an optimal assignment. Define a partition of the set of colors based on their indices, as follows:

- $\mathcal{C}' = \{j \mid B(j) = OPT(j)\}$; i.e., color indices for which the assignment made by algorithm **Balance** coincides with (that of) the optimal solution.
- $\mathcal{C}'' = \{0, \dots, m-1\} \setminus \mathcal{C}'$; i.e., colors indices for which the assignment made by algorithm **Balance** is different from the one of the optimal solution.

Lemma 21. *Assume \mathcal{C}'' is not empty (for otherwise the assignment computed by **Balance** would be optimal) and let $j \in \mathcal{C}''$. Let $k \neq j$ be any other color index in \mathcal{C}'' such that $B(k) = OPT(j)$. Then*

$$Q_{j,OPT(j)} \leq \max\{2 \cdot Q_{j,B(j)}, Q_{k,B(k)}\}.$$

Proof. First observe that, as $j, k \in \mathcal{C}''$ and $B(k) = \text{OPT}(j) \neq B(j)$, we have that $B(j) \neq B(k)$.

If $Q_{j, \text{OPT}(j)} \leq Q_{k, B(k)}$ we are clearly done. Suppose now that $Q_{j, \text{OPT}(j)} > Q_{k, B(k)}$, then we can prove that $Q_{j, \text{OPT}(j)} \leq 2 \cdot Q_{j, B(j)}$.

The fact that $j \in \mathcal{C}''$ means that **Balance** assigned color c_j to a different agent compared to the assignment of the optimal solution. Let r be the stage of **Balance** execution in which agent $\text{OPT}(j)$ processed color c_j (i.e., $Q_{j, \text{OPT}(j)} \in I_r$) and could not self assign c_j , then (in principle) one of the following conditions was true at stage r :

1. $\text{OPT}(j)$ already reached its maximum number of colors before stage r .

However, this is impossible. It is in fact a contradiction that $\text{OPT}(j)$ gets color c_k (recall that $\text{OPT}(j) = B(k)$) but does not get color c_j under **Balance**, since we are assuming $Q_{j, \text{OPT}(j)} > Q_{k, \text{OPT}(j)}$, which means that the assignment of c_k cannot be done earlier than c_j 's assignment.

2. Color c_j has already been assigned to $B(j)$. This might happen because

- (a) c_j has been assigned to $B(j)$ in a previous stage.

This implies that $B(j)$ has a larger number of items of color c_j with respect to $\text{OPT}(j)$, i.e., that $Q_{j, \text{OPT}(j)} \leq Q_{j, B(j)} \leq 2 \cdot Q_{j, B(j)}$.

- (b) c_j has been assigned to $B(j)$ in the same stage, because it has a smaller label on the ring.

By Lemma 20 we then have that $Q_{j, \text{OPT}(j)} \leq 2 \cdot Q_{j, B(j)}$.

□

Theorem 22. *Balance is a 3-approximation algorithm for the Distributed Balanced Color Assignment Problem.*

Proof. Let Cost_B and Cost_{OPT} be the cost of the solutions given by algorithm **Balance** and OPT , respectively. We can express these costs in the following way (where, for simplicity, we omit index i 's range, that is always $[0, n - 1]$):

$$\begin{aligned}
\text{Cost}_B &= \sum_{j=0}^{m-1} \sum_{i \neq B(j)} Q_{j,i} \\
&= \sum_{j \in \mathcal{C}'} \sum_{i \neq B(j)} Q_{j,i} + \sum_{j \in \mathcal{C}''} \sum_{i \neq B(j)} Q_{j,i} \\
&= \sum_{j \in \mathcal{C}'} \sum_{i \neq B(j)} Q_{j,i} + \sum_{j \in \mathcal{C}''} \left(Q_{j, \text{OPT}(j)} + \sum_{\substack{i \neq \text{OPT}(j), \\ i \neq B(j)}} Q_{j,i} \right).
\end{aligned}$$

Analogously,

$$Cost_{OPT} = \sum_{j \in \mathcal{C}'} \sum_{i \neq OPT(j)} Q_{j,i} + \sum_{j \in \mathcal{C}''} \left(Q_{j,B(j)} + \sum_{\substack{i \neq OPT(j), \\ i \neq B(j)}} Q_{j,i} \right)$$

By definition, $B(j) = OPT(j)$, for $j \in \mathcal{C}'$, and thus $\sum_{j \in \mathcal{C}'} \sum_{i \neq B(j)} Q_{j,i} = \sum_{j \in \mathcal{C}'} \sum_{i \neq OPT(j)} Q_{j,i}$, i.e., the cost associated with color $c_j \in \mathcal{C}'$ is exactly the same for **Balance** and *OPT*. Notice also that the term $\sum_{j \in \mathcal{C}''} \sum_{\substack{i \neq OPT(j), \\ i \neq B(j)}} Q_{j,i}$ appears in both cost expressions. Hence, to prove that $Cost_B \leq 3 \cdot Cost_{OPT}$, it is sufficient to show that

$$\sum_{j \in \mathcal{C}''} Q_{j,OPT(j)} \leq 3 \sum_{j \in \mathcal{C}''} Q_{j,B(j)}. \quad (4)$$

We can assume without loss of generality that m is a multiple of n . Indeed, if otherwise n does not divide m , we can add r dummy colors (for $r = m - \lfloor m/n \rfloor$), i.e. such that $Q_{j,i} = 0$ for all agents i and dummy color j . Since in our algorithm the agents consider the weights in decreasing order, the dummy colors will be processed at the end and therefore they have no effect on the assignment of the other colors. Moreover, as their weights are zero, they do not cause any change in the cost of the solution.

To prove (4), we build a partition of the set \mathcal{C}'' according to the following procedure. We start from any j_1 in \mathcal{C}'' and find another index j_2 such that $B(j_2) = OPT(j')$ for some $j' \in \mathcal{C}'' \setminus \{j_2\}$. Note that, since m is a multiple of n , every agent must have m/n colors and therefore such an index j_2 must exist. If $j' = j_1$ the procedure ends, otherwise we have found another index $j_3 = j'$ such that $B(j_3) = OPT(j'')$. Again, if $j'' = j_1$ the procedure ends, otherwise we repeat until, for some $t \geq 2$, we eventually get $B(j_t) = OPT(j_1)$. We then set

$$\mathcal{C}_1 = \{(j_1, j_2), (j_2, j_3), \dots, (j_{t-1}, j_t)\}.$$

If during this procedure we considered all indices in \mathcal{C}'' we stop, otherwise, we pick another index not appearing in \mathcal{C}_1 and repeat the same procedure to define a second set \mathcal{C}_2 , and so on until each index of \mathcal{C}'' appears in one \mathcal{C}_i . Observe that each \mathcal{C}_i contains at least two pairs of indices and that each index $j \in \mathcal{C}''$ appears in exactly two pairs of exactly one \mathcal{C}_i .

Then, using Lemma 21, we get

$$\begin{aligned}
\sum_{j \in \mathcal{C}''} Q_{j, \text{OPT}(j)} &= \sum_{\mathcal{C}_i} \sum_{(j, j') \in \mathcal{C}_i} Q_{j, \text{OPT}(j)} \\
&\leq \sum_{\mathcal{C}_i} \sum_{(j, j') \in \mathcal{C}_i} \max\{2 \cdot Q_{j, \mathbf{B}(j)}, Q_{j', \mathbf{B}(j')}\} \\
&\leq \sum_{\mathcal{C}_i} \sum_{(j, j') \in \mathcal{C}_i} (2 \cdot Q_{j, \mathbf{B}(j)} + Q_{j', \mathbf{B}(j')}) \\
&= \sum_{j \in \mathcal{C}''} 3Q_{j, \mathbf{B}(j)}
\end{aligned}$$

□

The following theorem shows that the approximation factor given in Theorem 5 is tight.

Theorem 23. *For any $0 < \epsilon < 1$, there exist instances of the Balanced Color Assignment Problem such that COST_B is a factor $3 - 4\epsilon/(4\delta + \epsilon)$ larger than the optimal cost, for some $0 < \delta < 1$.*

Proof. Consider the following instance of the balanced color assignment problem. For the sake of presentation, we assume that $m = n$ and that n is even, but it is straightforward to extend the proof to the general case.

Fix any rational $\epsilon > 0$, and let $q, \delta > 0$ be such that $q\epsilon/4$ is an integer and $q\delta = \lfloor q \rfloor$. Consider an instance of the problem such that colors are distributed as follows:

$$\begin{cases} Q_{2i, 2i} &= q(\delta + \epsilon/4) \\ Q_{2i+1, 2i} &= q \\ Q_{2i, 2i+1} &= q(2\delta - \epsilon/4) \\ Q_{2i+1, 2i+1} &= 0, \end{cases} \quad i = 0, 1, \dots, \frac{n}{2} - 1$$

and that a_0 is the leader elected in the first stage of algorithm **Balance**, and that the labels assigned to agents a_1, \dots, a_{n-1} are $1, \dots, n-1$, respectively.

Consider agents a_{2i} and a_{2i+1} , for any $0 \leq i \leq \frac{n}{2} - 1$. We can always assume that q is such that

$$\frac{\hat{p}}{2^{r+1}} \leq q\delta < q(\delta + \epsilon/4) < q(2\delta - \epsilon/4) < \frac{\hat{p}}{2^r},$$

for some r . That is, the weights of color c_{2i} for agents a_{2i} and a_{2i+1} belong to the same interval $[\hat{p}/2^{r+1}, \hat{p}/2^r)$.

It is easy to see that the optimal assignment gives c_{2i+1} to a_{2i} and c_{2i} to a_{2i+1} . The corresponding cost is $\text{Cost}_{\text{OPT}} = \frac{n}{2}q(\delta + \epsilon/4)$. On the other hand, algorithm **Balance** assigns c_{2i} to a_{2i} and c_{2i+1} to a_{2i+1} , with a corresponding cost $\text{Cost}_B = \frac{n}{2}q(3\delta - \epsilon/4)$. Hence, for the approximation factor, we get

$$\frac{Cost_B}{Cost_{OPT}} = \frac{3\delta - \frac{\epsilon}{4}}{\delta + \frac{\epsilon}{4}} = \frac{3(\delta + \frac{\epsilon}{4})}{\delta + \frac{\epsilon}{4}} - \frac{\frac{3\epsilon}{4} + \frac{\epsilon}{4}}{\delta + \frac{\epsilon}{4}} = 3 - \frac{4\epsilon}{4\delta + \epsilon}.$$

□

Even if the approximability result is tight, if we are willing to pay something in message complexity, we can get a 2-approximation algorithm.

Corollary 24. *Algorithm Balance can be transformed into a 2-approximation algorithm, by paying an additional multiplicative $O(\log p)$ factor in message complexity.*

Proof. Algorithm Balance is modified in the following way: colors in stage r of STEP 2 in Phase 3 are assigned to the agent having the largest number of items (falling in the interval I_r) and not to the one close to the leader. This can be achieved by making the agent forward on the ring, not only their choice of colors, but also their $Q_{i,j}$ s for those colors. This requires extra $O(\log p)$ bits per color, increasing total message complexity of such a multiplicative factor.

For what concerns the approximation factor, this modification to the algorithm allows to restate the thesis of Lemma 20 without the 2 multiplicative factor and, following the same reasoning of Theorem 5, conclude the proof.

□

Finally, if we are not willing to pay extra message complexity, but we are allowed to wait for a longer time, we get a $(2 + \epsilon)$ -approximation algorithm.

Theorem 25. *Assuming $m \in O(n^c)$, for some constant c , for any $0 < \epsilon < 1$, there is a $(2 + \epsilon)$ -approximation algorithm for the Distributed Balanced Color Assignment Problem with running time $O(n \log_{1+\epsilon} p)$ and message complexity $\Omega(nm)$.*

Proof. Modify the two interval threshold values of algorithm Sync-Balance in the following way:

$$l_r = \left\{ \frac{\hat{p}}{(1 + \epsilon)^{r+1}} \right\} \text{ and } u_r = \left\{ \frac{\hat{p}}{(1 + \epsilon)^r} \right\},$$

and redefine

$$\left\{ \frac{a}{b} \right\} = \begin{cases} \left\lceil \frac{a}{b} \right\rceil & \text{if } \frac{a}{b} > \frac{1}{1+\epsilon}; \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Accordingly, the statement of Lemma 20 becomes $Q_{j,i} \leq (1 + \epsilon)Q_{j,k}$, and the statement of Lemma 21 can be rewritten as

$$Q_{j,OPT(j)} \leq \max\{(1 + \epsilon) \cdot Q_{j,B(j)}, Q_{k,B(k)}\}.$$

The result on the approximation factor then follows by the same arguments of the proof of Theorem 5. The message complexity is not affected by these changes, while the running time now depends on the number of stages in Phase 3, that is $O(\log_{1+\epsilon} p)$. □

6 Conclusion

In this paper we have considered the Distributed Balanced Color Assignment problem, which we showed to be the distributed version of different matching problems. In the distributed setting, the problem models situations where agents search a common space and need to rearrange or organize the retrieved data.

Our results indicate that these kinds of problems can be solved quite efficiently on a ring, and that the loss incurred by the lack of centralized control is not significant. We have focused our attention to distributed solutions tailored for a ring of agents. A natural extension would be to consider different topologies and analyze how our techniques and ideas have to be modified in order to give efficient algorithms in more general settings. We believe that the main ideas contained in this work could be useful to extend the results even to arbitrary topologies. Indeed, an $O(n \text{ polylog}(n))$ distributed leader election protocol (that is needed in our algorithm) is also available for arbitrary ad hoc radio networks [5].

For what concerns the ring topology, it is very interesting to note that the value p never appears in the message complexity for the synchronous case (not even if the polynomial relation between m and n does not hold), while a factor $\log p$ appears in the asynchronous case. It is still an open question if it is possible to devise an asynchronous algorithm that shows optimal message complexity, under the same hypothesis of the synchronous one; *i.e.*, if it is possible to eliminate the extra $\log p / \log n$ factor.

Acknowledgments. The authors wish to thank Bruno Codenotti for many helpful comments and discussions.

References

- [1] Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, A. Keren, An Opportunity Cost Approach for Job Assignment and Reassignment in a Scalable Computing Cluster, *IEEE Trans. on Parallel and Distributed Systems*, 11 (2000) 760-768 .
- [2] Handbook of Computational Economics, edited by K. Schmedders, K. Judd, NORTH-HOLLAND (2013).
- [3] G. J. Chang, P. H. Ho, The β -assignment Problems, *European J. Oper. Research* 104 (1998) 593-600.
- [4] R. S. Chang, R. C. T. Lee, On a scheduling problem where a job can be executed only by a limited number of processors, *Computers and Operation Research*, 15 (1988) 471-478.
- [5] B. S. Chlebus, D. R. Kowalski, A. Pelc, Electing a Leader in Multi-hop Radio Networks, *Proceedings of Principles of Distributed Systems (7702) LNCS, pp 106-120, OPODIS 2012, Rome, Italy*.
- [6] G. De Marco, M. Leoncini, M. Montangero, Distributed algorithm for a color assignment on asynchronous rings, *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece*.

- [7] R. G. Gallager, P. A. Humblet, P. M. Spira, A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems* 5(1), 6677 (1983).
- [8] M. Hanckowiak, M. Karonski, A. Panconesi, On the distributed complexity of computing maximal matchings, *Proc. of SODA 98, the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, (1998) 219-225.
- [9] M. Hanckowiak, M. Karonski, A. Panconesi, A faster distributed algorithm for computing maximal matchings deterministically, *Proc. of PODC 99, the Eighteenth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, (1999) 219-228.
- [10] J. E. Hopcroft, R. M. Karp, An $n^{\frac{5}{2}}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM J. Comput.*, 2 (1973) 225-231.
- [11] D. S. Hirschberg, J. B. Sinclair, Decentralized extrema-finding in circular configurations of processes, *Communication of the ACM*, 23 (11) (1980) 627-628.
- [12] D. R. Kowalski, A. Pelc, Leader election in ad hoc radio networks: A keen ear helps, *In Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP), Part II. pp. 521533. LNCS 5556 (2009)*.
- [13] H. W. Kuhn, The Hungarian Method for the Assignment Problem, *Naval Research Logistics Quarterly*, 2 (1955), 83-97.
- [14] H. Kavalionak, E. Carlini, L. Ricci, A. Montresor, M. Coppola, Integrating peer-to-peer and cloud computing for massively multiuser online games, *Peer-to-Peer Networking and Applications*, (2013) 1-19.
- [15] J. Könemann, R. Ravi, A Matter of Degree: Improved Approximation Algorithms for Degree-Bounded Minimum Spanning Trees, *SIAM J. on Computing*, 31 (6) (2002) 1645-1958.
- [16] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, 1996.
- [17] L. Lovasz, M. D. Plummer, Matching Theory, *Annals of Discrete Mathematics* 29; North-Holland Mathematics Studies 121, (1986).
- [18] D. Peleg, Distributed Computing: A Locality-Sensitive Approach, *SIAM*, Philadelphia, PA, 2000.
- [19] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, S. Haridi, CLive: Cloud-assisted P2P live streaming, *Proc. of IEEE 12th International Conference on Peer-to-Peer Computing*, P2P'12, (2012) 79-90.
- [20] B. Schieber and S. Moran, Parallel Algorithms for Maximum Bipartite Matching and Maximum 0-1 Flows, *J. of Parallel and Distributed Computing*, 6 (1989), 20-38.

- [21] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *SIGCOMM Comput. Commun. Rev., Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 31 (4) (2001) 149-160.
- [22] J. Šilk, B. Robič, Processor Allocation Based on Weighted Bipartite Matching Scheduling, *Technical Report CSD-97-1, Computer System Department, Jožef Stefan Institut*, Ljubljana, Slovenia.